IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

Diajukan sebagai syarat menyelesaikan mata kuliah Strategi Algoritma (IF2211) Kelas RC
di Program Studi Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Sumatera

Oleh: Kelompok 4 (KoPiYah)

Daniel Calvin Simanjuntak	123140004
Danang Ridho Laksono	123140005
Ar Rauf Setiawan MJ	123140032

Dosen Pengampu: Winda Yulita, M.Cs.

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

1.1 Latar Belakang

"Diamonds" adalah sebuah programming challenge yang mengadu kecerdasan buatan dalam bentuk bot yang bertujuan mengumpulkan diamond sebanyak mungkin di arena permainan. Permainan ini berlangsung di atas papan dengan ukuran yang dapat disesuaikan, di mana bot pemain memulai dari base acak dan berusaha mengumpulkan diamond biru (1 poin) dan merah (2 poin). Poin baru dihitung setelah diamond dalam inventory berhasil dibawa kembali ke base. Tantangan semakin kompleks dengan adanya berbagai rintangan, objek seperti teleporter, tombol pengacak diamond, serta kemampuan bot untuk bergerak satu petak (horizontal/vertikal) dan melakukan tackle terhadap bot lawan, yang mengakibatkan bot lawan kembali ke base dan kehilangan diamond di inventory-nya.

Untuk meraih kemenangan, setiap pemain dituntut untuk merancang dan mengimplementasikan strategi algoritma yang efektif pada bot mereka. Mengingat kompleksitas dan sifat dinamis permainan "Diamonds", Tugas Besar mata kuliah Strategi Algoritma (IF2211) tahun 2025 ini berfokus pada penerapan Algoritma Greedy. Pendekatan greedy diharapkan dapat memandu bot dalam membuat keputusan optimal secara lokal pada setiap langkah, seperti memilih diamond terdekat atau bernilai tinggi, dengan harapan mencapai perolehan skor maksimum dalam satu ronde permainan.

1.2 Tujuan Tugas Besar

Tujuan utama dari Tugas Besar IF2211 ini adalah sebagai berikut:

1. Menerapkan konsep algoritma greedy untuk merancang dan mengimplementasikan strategi pada sebuah agen (bot) dalam permainan "Diamonds".
2. Mengembangkan program bot yang mampu berinteraksi dengan game engine dan secara otonom membuat keputusan berdasarkan strategi greedy yang telah dirancang untuk memaksimalkan perolehan diamond.
3. Menganalisis dan membandingkan berbagai alternatif pendekatan greedy untuk menentukan solusi yang dianggap paling optimal dalam konteks permainan.
4. Mendokumentasikan proses perancangan, implementasi, dan pengujian algoritma dalam sebuah laporan ilmiah.

5. Menguji dan mengkompetisikan bot yang telah dikembangkan dengan bot dari kelompok lain untuk mengevaluasi kinerja strategi yang diimplementasikan ("Rilis Tubes Stima 2025 (1).pptx", Slide 10).

1.3 Ruang Lingkup Tugas

Ruang lingkup pengerjaan Tugas Besar ini meliputi:

1. Pemahaman terhadap aturan permainan "Diamonds", mekanisme interaksi antara bot dan game engine, serta struktur data yang digunakan.
2. Penggunaan **Game Engine** yang disediakan, yang bertanggung jawab atas logika inti permainan dan penyediaan API untuk komunikasi.
3. Modifikasi dan pengembangan **Bot Starter Pack** yang disediakan, khususnya pada bagian logika bot (game/logic/), untuk mengimplementasikan satu atau lebih strategi algoritma greedy.
4. Perancangan algoritma greedy yang mempertimbangkan berbagai aspek permainan seperti posisi diamond, nilai diamond, rintangan, dan kemungkinan keberadaan teleporter.
5. Implementasi algoritma terpilih ke dalam bahasa pemrograman Python, sesuai dengan kerangka Bot Starter Pack.
6. Pengujian fungsionalitas dan efektivitas bot dalam mengumpulkan diamond.
7. Penyusunan laporan Tugas Besar yang mencakup deskripsi tugas, landasan teori, desain dan implementasi algoritma, serta analisis hasil.

Tugas ini tidak mencakup modifikasi pada Game Engine. Fokus utama adalah pada perancangan dan implementasi logika bot.

1.4 Spesifikasi Tugas

Berikut adalah spesifikasi rinci terkait pelaksanaan dan penilaian Tugas Besar ini:

1. **Pengerjaan Kelompok:** Tugas dikerjakan secara berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang.
2. **Komponen Utama:**
    - **Game Engine:** Disediakan oleh tim asisten, berisi backend (logika game, API) dan frontend (visualisasi)
      Link :https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0.
    - **Bot Starter Pack:** Disediakan oleh tim asisten, berisi kerangka program untuk memanggil API dan logika bot yang akan diimplementasikan oleh mahasiswa.
      Link : https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1.

3. **Algoritma:** Wajib menggunakan strategi algoritma **Greedy**.
4. **Luaran (Deliverables):**
    - Program bot yang fungsional.
    - Laporan Tugas Besar dalam format yang ditentukan.
5. **Pengumpulan Laporan:** Laporan dikumpulkan pada hari Minggu, 1 Juni 2025, paling lambat pukul 23.59 melalui Google Form yang ditentukan..
6. **Penilaian ("Rilis Tubes Stima 2025 (1).pptx", Slide 12):**
    - Desain Solusi Algoritma Greedy yang ditulis dalam Laporan: 45%
    - Implementasi Program dan Demo: 55%
7. **Kompetisi:** Bot yang telah dibuat akan dikompetisikan. Terdapat hadiah bagi pemenang .
8. **Integritas Akademik:** Dilarang keras melakukan *copy-paste* program dari internet (AI, repositori lain, ataupun teman). Pelanggaran akan mengakibatkan nilai nol. Program harus dapat dijalankan setidaknya pada sistem operasi Windows dan Linux. Program yang tidak dapat dijalankan tidak akan dinilai .

BAB II
LANDASAN TEORI

2.1 Dasar Teori

2.1.1 Algoritma Greedy

Algoritma greedy adalah salah satu paradigma perancangan algoritma yang populer untuk menyelesaikan masalah optimasi. Prinsip dasar dari algoritma greedy adalah membuat serangkaian pilihan yang tampak optimal secara lokal pada setiap langkah dengan harapan akan mengarah pada solusi optimal global [1]. Pada setiap tahap pengambilan keputusan, algoritma greedy memilih opsi yang memberikan keuntungan terbesar atau biaya terkecil saat itu, tanpa mempertimbangkan konsekuensi pilihan tersebut terhadap keputusan di masa depan [2]. Harapannya, serangkaian pilihan lokal optimal ini akan mengarah pada solusi global optimal atau setidaknya solusi yang cukup baik. Dalam konteks permainan 'Diamonds', ini berarti bot akan membuat keputusan pergerakan berdasarkan kriteria tertentu—misalnya, bergerak ke diamond terdekat dengan nilai tertinggi—untuk memaksimalkan perolehan skor pada setiap kesempatan yang ada, sesuai dengan tujuan utama yaitu mengumpulkan diamond sebanyak-banyaknya.

Karakteristik utama dari algoritma greedy meliputi:

1. **Prinsip Greedy Choice Property**: Solusi optimal global dapat dicapai dengan membuat pilihan optimal secara lokal. Artinya, pilihan yang dibuat pada setiap langkah tidak tergantung pada pilihan sebelumnya atau pilihan di masa depan, melainkan hanya pada kondisi saat itu.
2. **Optimal Substructure**: Solusi optimal untuk suatu masalah mengandung solusi optimal untuk sub-masalahnya. Jika sebuah pilihan optimal lokal dibuat, masalah yang tersisa juga harus memiliki solusi optimal setelah pilihan tersebut diterapkan [3].

Meskipun algoritma greedy seringkali intuitif dan mudah diimplementasikan, tidak semua masalah optimasi dapat diselesaikan secara optimal menggunakan pendekatan ini. Algoritma greedy dapat menghasilkan solusi suboptimal atau bahkan salah untuk beberapa jenis masalah. Keberhasilan algoritma greedy sangat bergantung pada struktur masalah yang dihadapi.

Beberapa masalah klasik yang dapat diselesaikan secara optimal dengan algoritma greedy antara lain adalah masalah penukaran uang dengan koin seminimal mungkin (untuk denominasi tertentu), algoritma Prim dan Kruskal untuk mencari Minimum Spanning Tree (MST), dan algoritma Dijkstra untuk mencari lintasan terpendek [4].

2.1.2 Permainan Multi-Agen dan Pengambilan Keputusan

Dalam konteks permainan seperti "Diamonds", di mana terdapat lebih dari satu bot yang berinteraksi dalam satu lingkungan, konsep sistem multi-agen (Multi-Agent System - MAS) menjadi relevan. Setiap bot dapat dianggap sebagai agen otonom yang bertujuan untuk memaksimalkan skornya sendiri [5]. Pengambilan keputusan dalam lingkungan seperti ini seringkali melibatkan ketidakpastian karena aksi agen lain dapat mempengaruhi keadaan permainan dan hasil dari pilihan yang dibuat oleh agen kita. Strategi greedy dalam konteks ini berarti bot akan membuat keputusan yang paling menguntungkan bagi dirinya pada saat itu, misalnya bergerak ke diamond terdekat atau diamond dengan nilai tertinggi, dengan mempertimbangkan informasi yang tersedia saat itu [6].

2.1.3 Pathfinding (Pencarian Jalur)

Dalam permainan "Diamonds", kemampuan bot untuk menavigasi peta dan menemukan jalur menuju diamond adalah krusial. Algoritma pencarian jalur seperti Breadth-First Search (BFS) atau Dijkstra sering digunakan. BFS dapat menemukan jalur terpendek dalam graf tak berbobot, yang cocok jika setiap langkah gerakan memiliki biaya yang sama. Algoritma Dijkstra, di sisi lain, dapat menangani graf berbobot, di mana biaya antar node (misalnya, sel pada peta) bisa berbeda [4], [7]. Pendekatan greedy dalam pathfinding dapat berupa pemilihan langkah berikutnya yang secara heuristik paling mendekati target.

2.1.4 Minimum Spanning Tree (MST)

Algoritma seperti Kruskal dan Prim digunakan untuk menemukan Minimum Spanning Tree (MST) dalam sebuah graf. MST adalah subgraf yang menghubungkan semua node dalam graf tanpa membentuk siklus dan dengan total bobot edge minimum [3]. Dalam konteks permainan "Diamonds", MST bisa secara konseptual digunakan untuk merencanakan urutan pengambilan diamond secara efisien, di mana diamond dianggap sebagai node dan jarak antar

diamond sebagai bobot edge. Bot kemudian dapat mengikuti jalur yang terbentuk dari MST tersebut, meskipun aplikasi langsungnya mungkin memerlukan adaptasi karena sifat dinamis permainan [8].

2.2 Cara Kerja Program

2.2.1 Prasyarat dan Instalasi Lingkungan

Sebelum memulai proses pengembangan dan eksekusi permainan "Diamonds", terdapat serangkaian prasyarat perangkat lunak yang harus dipenuhi oleh pengguna. Setelah pra syarat tersebut terpenuhi, pengguna dapat melanjutkan ke tahap instalasi dan konfigurasi awal game engine.

Untuk memastikan kelancaran, sistem pengguna harus telah terinstal dengan **Node.js**, yang berfungsi sebagai lingkungan eksekusi JavaScript esensial untuk game engine; tautan unduhan tersedia di situs resmi Node.js. Selanjutnya, diperlukan **Docker Desktop**, sebuah platform untuk virtualisasi dan manajemen kontainer yang akan digunakan untuk menjalankan database lokal permainan; ini dapat diunduh dari situs Docker. Terakhir, **Yarn** sebagai manajer paket untuk Node.js juga dibutuhkan. Yarn dapat diinstal secara global setelah Node.js terpasang, biasanya melalui Node Package Manager (npm) dengan menjalankan perintah npm install --global yarn di terminal.

Setelah semua prasyarat perangkat lunak terpenuhi, proses instalasi dan konfigurasi awal game engine dapat dimulai. Langkah pertama adalah mengunduh kode sumber game engine, biasanya dalam format .zip, dari rilis resmi yang disediakan, dan kemudian mengekstraknya ke direktori lokal pada komputer pengguna. Setelah itu, pengguna perlu membuka terminal atau command prompt, masuk ke dalam direktori root proyek game engine yang baru saja diekstrak, dan menjalankan perintah yarn untuk menginstal semua dependensi proyek yang dibutuhkan oleh game engine. Langkah berikutnya adalah melakukan setup variabel lingkungan default dengan menjalankan skrip yang tersedia di dalam subdirektori scripts/ proyek; untuk pengguna Windows, skrip yang dijalankan adalah ./copy-env.bat, sedangkan untuk pengguna Linux atau macOS, perintahnya adalah chmod +x ./scripts/copy-env.sh diikuti dengan ./copy-env.sh untuk

memberikan izin eksekusi dan menjalankan skrip tersebut. Tahap persiapan lingkungan dilanjutkan dengan setup database lokal. Ini memerlukan Docker Desktop yang sudah berjalan. Dari direktori root game engine, perintah docker compose up -d database dijalankan di terminal untuk menyiapkan dan menjalankan kontainer database. Setelah kontainer database aktif, skema database perlu diinisialisasi menggunakan Prisma dengan menjalankan skrip yang juga berada di direktori scripts/: ./setup-db-prisma.bat untuk Windows, atau chmod +x ./scripts/setup-db-prisma.sh && ./scripts/setup-db-prisma.sh untuk Linux/macOS. Jika diperlukan interaksi langsung dengan database untuk tujuan debugging atau pengelolaan lebih lanjut, pengguna dapat merujuk ke panduan "Get Started with Diamonds.pdf" untuk instruksi koneksi menggunakan psql di dalam kontainer Docker.

2.2.2 Alur Interaksi

Alur interaksi standar diawali dengan upaya program bot untuk memverifikasi eksistensinya atau melakukan pemulihan sesi. Ini dicapai dengan mengirimkan sebuah *request* HTTP POST ke *endpoint* /api/bots/recover. *Payload* dari *request* ini umumnya mencakup kredensial bot, seperti alamat email dan kata sandi. Respon positif dari *backend*, ditandai dengan kode status 200 OK, akan menyertakan id unik bot sebagai konfirmasi bahwa bot telah terdaftar sebelumnya. Sebaliknya, jika bot belum terdaftar, *backend* akan merespons dengan kode status 404 Not Found.

Menindaklanjuti respon 404 Not Found dari tahap pemulihan, program bot akan melanjutkan ke proses registrasi baru. Sebuah *request* HTTP POST kemudian dikirimkan ke *endpoint* /api/bots. *Payload* untuk registrasi ini umumnya terdiri dari detail lengkap bot, seperti email, name (nama tampilan bot), password untuk autentikasi, dan team (afiliasi tim). Keberhasilan registrasi akan dikonfirmasi oleh *backend* melalui *response code* 200 OK, yang juga akan menyertakan id unik untuk bot yang baru dibuat tersebut.

Setelah id bot berhasil diperoleh, baik melalui pemulihan sesi maupun registrasi baru, langkah selanjutnya adalah mengintegrasikan bot ke dalam sebuah sesi permainan aktif. Bot mengirimkan *request* HTTP POST ke *endpoint* dinamis /api/bots/{id}/join, di mana {id} adalah ID unik bot yang bersangkutan. *Payload request* ini biasanya berisi preferredBoardId, yang mengindikasikan papan permainan spesifik yang ingin dimasuki oleh bot. Jika proses bergabung berhasil dan papan permainan tersedia, *backend* akan merespons dengan kode 200 OK, disertai dengan *body* yang berisi informasi komprehensif mengenai kondisi awal papan permainan tersebut, yang seringkali direpresentasikan sebagai objek GameState.

Dengan bot yang telah berhasil bergabung ke dalam papan permainan, dimulailah siklus inti interaksi permainan. Berdasarkan GameState yang diterima dan logika strategi internalnya, program bot akan secara periodik menghitung dan menentukan gerakan berikutnya. Keputusan gerakan ini dikomunikasikan ke *backend* melalui *request* HTTP POST ke *endpoint* /api/bots/{id}/move. *Payload* dari *request* ini membawa data direction, yang menyatakan arah pergerakan bot (misalnya, "NORTH", "SOUTH", "EAST", "WEST", atau representasi arah lainnya sesuai definisi API). Setiap *request* gerakan yang valid dan berhasil diproses oleh *backend* akan direspons dengan kode 200 OK, disertai *body* yang berisi GameState terbaru setelah gerakan tersebut dieksekusi. Siklus pengiriman perintah gerakan ini berlanjut secara iteratif selama bot masih aktif dalam permainan dan durasi permainan belum berakhir. Penting untuk dicatat bahwa jika waktu partisipasi bot dalam satu sesi permainan telah habis, game engine akan secara otomatis mengeluarkan bot tersebut dari papan permainan.

Sementara itu, untuk memastikan representasi visual permainan tetap sinkron dengan kondisi aktual di sisi server, antarmuka pengguna grafis (*frontend*) yang diakses pemain melalui browser juga melakukan komunikasi periodik. *Frontend* akan mengirimkan *request* HTTP GET ke *endpoint* /api/boards/{id} (di mana {id} adalah ID papan permainan yang sedang ditampilkan) untuk mengambil GameState termutakhir. Ini memungkinkan visualisasi permainan untuk selalu merefleksikan perubahan yang terjadi akibat aksi semua bot dan dinamika permainan lainnya secara *real-time* atau mendekati *real-time*.

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Sebelum bot dapat membuat keputusan strategis, ia harus mampu memetakan dan memahami lingkungan permainannya. Proses mapping ini melibatkan interpretasi data yang diterima dari game engine pada setiap giliran. Informasi utama yang relevan untuk strategi greedy meliputi:

1. **Peta Permainan (game_state.map_info.map):** Representasi grid dari arena permainan. Setiap sel pada peta dapat berupa:
    - Area kosong yang dapat dilalui.
    - Dinding atau rintangan (OBSTACLE) yang tidak dapat dilalui.
    - Teleporter (TELEPORT) yang memindahkan bot ke titik lain.
    - Area jebakan/bahaya (jika ada dalam spesifikasi game).
2. **Posisi Diamond (game_state.map_info.diamonds):** Daftar koordinat (x, y) dari semua diamond yang tersedia di peta, beserta jumlah poin (points) yang dimiliki setiap diamond dan kemungkinan ID unik.
3. **Posisi Bot Sendiri (game_state.my_bot_info):** Koordinat (x, y) bot saat ini, jumlah diamond yang sudah dikumpulkan (score), sisa energi atau status lainnya (jika ada).
4. **Posisi Bot Lawan (game_state.other_bot_info):** Daftar informasi mengenai bot lawan, termasuk posisi mereka. Informasi ini penting untuk menghindari tabrakan atau memprediksi pergerakan lawan dalam strategi yang lebih lanjut, meskipun strategi greedy dasar mungkin tidak secara eksplisit menggunakannya secara kompleks.
5. **Informasi Tambahan:** Seperti game_state.map_info.teleporters yang berisi detail pasangan teleporter.

Data ini digunakan untuk membangun representasi internal dari keadaan permainan yang akan menjadi dasar bagi algoritma greedy dalam memilih tindakan.

3.2 Eksplorasi Alternatif Solusi Greedy

Beberapa pendekatan algoritma greedy dapat dipertimbangkan untuk permainan "Diamonds". Berikut adalah lima alternatif yang dieksplorasi:

3.2.1 Algoritma D’

Bot D’ akan menjalankan strategi greedy hibrida yang dilengkapi dengan sebuah mode operasional khusus yang disebut "Strategi Penghindaran dan Pengamanan Diamond Selektif". Aktivasi mode khusus ini menjadi pertimbangan awal bot. Bot akan masuk ke mode ini jika sisa waktu permainan berada dalam rentang tertentu (misalnya, STRATEGY_AVOID_AND_SAFE_DIAMOND_TIME_SECONDS detik sebelum permainan berakhir) DAN jumlah diamond yang saat ini dibawa oleh bot berada di antara STRATEGY MIN_DIAMONDS_FOR_AVOID dan STRATEGY_MAX_DIAMONDS_FOR_AVOID. Ketika dalam mode khusus ini, prioritas utama bot adalah kembali ke markas jika langkah tersebut aman dan memungkinkan untuk mengamankan diamond yang telah dibawa. Jika tidak kembali ke markas, bot hanya akan mencari dan menargetkan diamond yang berada dalam radius jarak aman dan layak tertentu dari posisinya saat ini (antara STRATEGY SAFE_DIAMOND_RADIUS_MIN dan STRATEGY_SAFE_DIAMOND_RADIUS_MAX). Selain itu, selama dalam mode ini, bot akan secara aktif berusaha menghindari bot lawan yang terdeteksi dalam radius STRATEGY_AVOID_OPPONENT_RADIUS, bahkan jika itu berarti harus mengabaikan diamond yang kurang ideal.

Jika bot tidak berada dalam Mode Strategi Defensif-Selektif, ia akan mengikuti serangkaian logika keputusan standar. Pertama, bot akan memeriksa riwayat posisinya (_position_history sepanjang HISTORY_LENGTH) untuk mendeteksi apakah ia terjebak. Jika terjebak (tidak bergerak atau bergerak siklik), targetnya akan dihapus dan bot akan dipaksa melakukan *roaming*. Selanjutnya, bot akan memprioritaskan kembali ke *base*. Keputusan ini akan diambil jika inventaris bot sudah dianggap penuh (misalnya, current_diamonds_held >= inventory_full_threshold), atau jika sisa waktu permainan sudah sangat kritis (kurang dari CRITICAL_TIME_RETURN_TO_BASE_SECONDS) dan bot sedang membawa diamond.

Apabila tidak ada kondisi untuk kembali ke *base*, bot akan mempertimbangkan penggunaan Tombol Merah. Ini dilakukan jika bot tidak membawa diamond, kondisi diamond di papan dinilai buruk (misalnya, menggunakan fungsi _are_diamonds_bad_or_unsafe untuk evaluasi), bot berada dalam jarak yang wajar dari Tombol Merah, dan periode *cooldown* sejak penggunaan terakhir telah terpenuhi.

Jika tidak ada prioritas di atas yang aktif, bot akan mencari diamond target terbaik menggunakan fungsi _find_best_diamond_target. Bot akan mengabaikan diamond yang sudah tercatat dalam _failed_targets (jika jumlah percobaan gagal telah melewati batas). Untuk diamond yang valid, bot akan menghitung skor berdasarkan rasio nilai poin terhadap jarak Manhattan (ditambah satu untuk menghindari pembagian dengan nol), dan juga mempertimbangkan keamanan diamond dari jangkauan bot lain (menggunakan fungsi seperti _is_diamond_safe_from_others dan _can_reach_before_others). Diamond dengan skor tertinggi yang memenuhi kriteria akan dipilih. Jika upaya mencapai diamond gagal berulang kali, diamond tersebut akan ditambahkan ke _failed_targets.

Sebagai langkah terakhir, jika tidak ada target yang valid atau pathfinding ke target gagal, bot akan melakukan *roaming* menggunakan pola arah yang telah ditentukan (ROAMING_DIRECTIONS). Pathfinding ke semua target menggunakan algoritma BFS standar.

3.2.2 Algoritma Garox

Bot Garox akan beroperasi dengan mengevaluasi serangkaian prioritas tindakan pada setiap giliran untuk menentukan langkah optimalnya. Prioritas utama Garox adalah kembali ke markas (*base*) jika kondisi tertentu terpenuhi. Bot akan memutuskan untuk pulang jika inventarisnya telah penuh dengan diamond, atau jika estimasi waktu yang dibutuhkan untuk perjalanan kembali ke markas (dengan memperhitungkan jalur paling efisien termasuk penggunaan teleporter dan ditambah margin keamanan waktu TIME_SAFETY_MARGIN_MOVES) sudah kritis relatif terhadap sisa waktu total permainan dan bot sedang membawa diamond. Selain itu, Garox akan memicu kondisi "pulang mendesak" jika persentase sisa waktu permainan sudah sangat rendah (misalnya, di bawah URGENT_TIME_PERCENTAGE) sementara bot membawa sejumlah diamond minimum tertentu (ditentukan oleh URGENT_RETURN_THRESHOLD_PERCENT dari kapasitas inventarisnya) dan waktu memang tidak memungkinkan untuk aksi lain. Jika Garox memutuskan

pulang dan sudah berada di markas namun masih ada kondisi pulang yang aktif (misalnya inventaris penuh), ia akan melakukan gerakan acak aman.

Jika tidak ada keharusan untuk pulang, Garox akan mengevaluasi diamond terbaik untuk diambil. Bot akan menganalisis setiap diamond yang ada di papan. Untuk masing-masing diamond, Garox akan menghitung jalur efektif (terpendek, termasuk opsi via semua teleporter yang ada) menuju diamond tersebut, dan juga jalur efektif dari diamond itu kembali ke markas. Total waktu untuk siklus ini (ambil diamond dan setor ke markas) akan diestimasi. Berdasarkan nilai diamond, estimasi total waktu, "faktor urgensi" yang meningkat seiring berkurangnya waktu permainan, dan "faktor keserakahan" yang meningkat jika inventaris masih kosong, Garox akan menghitung skor evaluasi komprehensif untuk setiap diamond. Diamond dengan skor tertinggi yang juga melampaui ambang batas minimal (BASE_MIN_TARGET_EVALUATION) akan menjadi target Garox. Bot akan secara eksplisit mengelola status jika targetnya adalah pintu masuk teleporter (current_target_is_teleporter_entry) untuk navigasi yang benar.

Apabila tidak ada diamond yang dinilai cukup menguntungkan, Garox akan mencari peluang untuk menyerang (melakukan *tackle*) bot lawan. Garox akan memindai area sekitarnya dalam radius TACKLE_RADIUS untuk mencari bot lawan yang membawa sejumlah diamond signifikan (minimal TACKLE_MIN_OPPONENT_DIAMONDS). Meskipun bersifat "full ofensif", Garox juga akan mempertimbangkan risiko terhadap dirinya jika ia juga sedang membawa banyak diamond, sebelum memutuskan untuk menyerang. Jika target *tackle* yang menjanjikan ditemukan, bot lawan tersebut akan menjadi tujuan Garox.

Jika ketiga prioritas di atas tidak menghasilkan target, Garox akan mempertimbangkan penggunaan Tombol Merah. Bot akan memilih opsi ini jika ia tidak sedang membawa diamond, jumlah diamond di papan di bawah ambang batas LOW_DIAMOND_COUNT_FOR_RED_BUTTON, bot berada cukup dekat dengan Tombol Merah (dengan mempertimbangkan RED_BUTTON_PROXIMITY_ADVANTAGE jika ada), dan periode *cooldown* (RED_BUTTON_COOLDOWN_TURNS) sejak penggunaan terakhir telah terpenuhi.

Sebagai pilihan terakhir, jika tidak ada aksi strategis yang bisa diambil, Garox akan melakukan gerakan *roaming* atau gerakan acak yang aman. Ini dilakukan untuk menghindari bot diam di tempat, terus menjelajahi peta, dan mendeteksi jika ia terjebak melalui mekanisme consecutive_identical_moves yang melebihi STUCK_MOVE_LIMIT.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

3.3.1 Algoritma D’

Kelebihan:

- **Adaptasi Perilaku Situasional:** Pengenalan mode strategi khusus memungkinkan bot mengubah perilakunya menjadi lebih defensif dan selektif pada periode permainan tertentu, yang dapat meningkatkan peluang mengamankan skor.
- **Manajemen Risiko yang Lebih Baik pada Fase Tertentu:** Mode khusus tersebut dirancang untuk mengurangi risiko kehilangan diamond ketika waktu mulai berharga.
- **Keseimbangan Strategi:** Mencoba menyeimbangkan antara pengumpulan diamond standar dengan fase permainan hati-hati.

Kekurangan:

- **Kompleksitas Tambahan:** Pengelolaan transisi masuk dan keluar dari mode strategi khusus serta logika di dalamnya menambah kompleksitas.
- ***Tuning* Parameter Mode Khusus:** Efektivitas mode strategi khusus sangat bergantung pada penyetelan parameter-parameter terkait (waktu, jumlah diamond, radius).
- **Potensi Kurang Agresif Secara Umum:** Di luar mode khusus, jika logika standarnya tidak seagresif Algoritma G, mungkin kehilangan beberapa peluang.
- **Tidak Ada Logika *Tackle* Ofensif:** Fokus utama tetap pada pengumpulan dan pengamanan, bukan konfrontasi langsung.

3.3.2 Algoritma Garox

Kelebihan:

- **Proaktif dan Oportunistik:** Mencari peluang untuk menyerang dan menggunakan Tombol Merah secara aktif.

- **Manajemen Waktu dan Risiko Pulang yang Canggih:** Mempertimbangkan sisa waktu permainan dan estimasi perjalanan untuk keputusan kembali ke *base*.
- **Penggunaan Teleporter Optimal:** Secara eksplisit menghitung dan memilih jalur tercepat, termasuk melalui teleporter.
- **Potensi Skor Tinggi:** Sifat agresifnya bisa menghasilkan skor tinggi jika berhasil memanfaatkan peluang.

Kekurangan:

- **Sangat Kompleks:** Implementasi logika dan *tuning* banyak parameter dan *threshold* sangat rumit.
- **Risiko Tinggi:** Sifat "full ofensif" dan pengurangan pengecekan defensif bisa membuat bot rentan jika strategi agresifnya gagal atau menemui lawan yang lebih cerdik.
- **Bergantung pada *Tuning* Parameter:** Kinerja sangat sensitif terhadap pengaturan banyak konstanta dan *threshold*.

Fitur/Aspek	Algoritma Garox	Algoritma D’
Prioritas Kembali ke Base	Sangat komprehensif: inventaris penuh, waktu kritis, urgensi akhir game, dekat base + isi.	Komprehensif: inventaris penuh, akhir game + isi, dekat base + isi.
Kesadaran Akhir Permainan	Ada (TIME_SAFETY_MARGIN_MOVES, URGENT_TIME_PERCENTAGE).	Ada (SAFE_TURN_LIMIT).
Logika Tombol Merah	Strategis: jika diamond sedikit/buruk, bot dekat, cooldown, tidak bawa diamond.	Strategis: jika diamond buruk/tidak aman (_are_diamonds_bad_or_unsafe), bot dekat, cooldown.

Pemilihan Diamond (Skoring & Pathing)	Skor kompleks (poin, waktu total ambil+pulang, urgensi, keserakahan), jarak efektif via BFS + teleporter.	Skor berdasarkan poin / (jarak_Manhattan + 1) + keamanan dari lawan, path via BFS.
Manajemen Target Gagal	Oportunistik, mungkin cepat beralih atau tidak ada detail eksplisit tentang counter kegagalan di penjelasan.	Kamus _failed_targets dengan counter kegagalan, menandai jika gagal berulang.
Deteksi & Penanganan Terjebak/Stuck	Ada (pergerakan acak aman jika tidak ada aksi, consecutive_identical_moves).	Ada (_position_history, _is_stuck), paksa roaming.
Interaksi Lawan (Tackle)	Proaktif: evaluasi target tackle jika menguntungkan.	Tidak ada logika tackle eksplisit. Fokus pada keamanan diamond sendiri.
Penggunaan Teleporter	Terintegrasi penuh dalam perhitungan jarak efektif dan navigasi.	Ditangani dalam BFS standar, mungkin tidak seoptimal Garox dalam pemilihan jalur via teleporter.
Sifat Umum	Agresif, oportunistik, kompleks, berisiko tinggi, potensi imbalan tinggi.	Terstruktur, manajemen risiko baik, adaptif terhadap kegagalan, seimbang.
Kompleksitas Implementasi & Tuning	Sangat Tinggi.	Tinggi (namun mungkin sedikit di bawah Garox).

Potensi Kinerja (dibanding satu sama lain)	Jauh Lebih Tinggi (jika diimplementasikan dan di-tune dengan benar, terutama karena aspek ofensif dan optimasi teleporter).	Baik hingga Sangat Baik (strategi solid, tapi kurangnya aspek ofensif bisa jadi batasan).

3.4 Strategi Greedy yang Dipilih

Setelah melakukan eksplorasi terhadap berbagai alternatif solusi algoritma greedy pada subbab 3.2, algoritma yang dipilih untuk implementasi detail dan dianggap memiliki potensi paling besar untuk kinerja optimal dalam permainan "Diamonds" yang kompleks dan dinamis adalah **"Algoritma Garox"**

Algoritma Garox dipilih karena pendekatan strategisnya yang komprehensif, agresif, dan oportunistik, yang secara teori memiliki potensi lebih besar untuk mencapai skor tinggi dalam permainan "Diamonds" yang kompetitif. Fitur-fitur unggulannya meliputi:

1. **Prioritas Keputusan Berlapis dan Kontekstual:** Garox tidak hanya mengandalkan satu metrik, tetapi mengevaluasi serangkaian kondisi secara hierarkis (kembali ke *base*, evaluasi diamond, serangan lawan, Tombol Merah, *roaming*), yang memungkinkan respons lebih adaptif.
2. **Manajemen Risiko dan Waktu yang Canggih:** Pertimbangan sisa waktu permainan, estimasi waktu perjalanan (termasuk teleporter), dan margin keamanan untuk keputusan kembali ke *base* menunjukkan pemikiran strategis jangka pendek dan menengah.
3. **Pemanfaatan Fitur Peta Secara Optimal:** Integrasi penuh penggunaan teleporter dalam perhitungan jarak efektif dan navigasi, serta penggunaan Tombol Merah yang strategis.
4. **Dimensi Ofensif:** Kemampuan untuk mengevaluasi dan melakukan serangan (*tackle*) pada lawan menambah dimensi strategi yang tidak dimiliki oleh algoritma yang lebih pasif.
5. **Potensi Mendominasi Permainan:** Sifatnya yang "full ofensif" dan oportunistik, jika berhasil, dapat mengganggu lawan sekaligus memaksimalkan perolehan poin.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

```
CLASS GaroxLogic EXTENDS BaseLogic

KAMUS // Atribut Kelas dan Parameter Konfigurasi (sesuai Tabel 1 di 3.3.1)
    // --- Atribut Status Dinamis Utama ---
    goal_position: Optional<Position>
    current_target_is_teleporter_entry: boolean
    position_history: List<Position>
    last_red_button_press_turn: integer
        teleporter_data_cache: Optional<List<Tuple<Position, Position, int>>> // Cache pasangan teleporter & jarak antar mereka
    consecutive_identical_moves: integer
    turns_since_last_diamond_scored: integer
    // ... (atribut dinamis lainnya jika ada, misal terkait state tackle khusus)

    // --- Parameter Konfigurasi Utama (diinisialisasi di Constructor) ---
    DEFAULT_INVENTORY_SIZE: integer
    TIME_SAFETY_MARGIN_MOVES: integer
    TACKLE_RADIUS: integer
    TACKLE_MIN_OPPONENT_DIAMONDS: integer
    OPPONENT_HIGH_DIAMOND_COUNT: integer // Mungkin digunakan untuk prioritas tackle
    LOW_DIAMOND_COUNT_FOR_RED_BUTTON: integer
    URGENT_TIME_PERCENTAGE: float
    NORMAL_RETURN_THRESHOLD_PERCENT: float
    URGENT_RETURN_THRESHOLD_PERCENT: float
    BASE_MIN_TARGET_EVALUATION: float // Skor evaluasi minimal untuk diamond
    STUCK_MOVE_LIMIT: integer // Batas gerakan identik sebelum dianggap stuck
    RED_BUTTON_COOLDOWN_TURNS: integer
    TIME_PER_STEP_MS_ESTIMATE: integer // Estimasi waktu per langkah bot
```

```
    MIN_TIME_GAIN_FOR_TP_VS_MANHATTAN: integer // Minimal penghematan waktu untuk memilih teleporter
    // ... (parameter konfigurasi lainnya)
ENDKAMUS

ALGORITMA

FUNCTION Constructor()
    // Inisialisasi semua atribut status dinamis ke nilai awal
    goal_position ← NIL
    current_target_is_teleporter_entry ← FALSE
    position_history ← EMPTY_LIST
        last_red_button_press_turn ← -(RED_BUTTON_COOLDOWN_TURNS + 5) // Agar bisa langsung dipakai
    teleporter_data_cache ← NIL
    consecutive_identical_moves ← 0
    turns_since_last_diamond_scored ← 0

    // Inisialisasi parameter konfigurasi dengan nilai-nilai yang sudah ditentukan
    // Contoh:
    DEFAULT_INVENTORY_SIZE ← 5
    TIME_SAFETY_MARGIN_MOVES ← 15
    TACKLE_RADIUS ← 1
    // ... dan seterusnya untuk semua parameter dari KAMUS ...
ENDFUNCTION

// ----------------------------------------------------------------
FUNCTION next_move(gameState: GameState) : Tuple[dx, dy]
    // Ekstrak informasi penting dari gameState
    myBotInfo ← CALL _get_game_info_from_state(gameState)
    // myBotInfo berisi: currentPos, diamondsHeld, basePos, score, inventorySize, timeLeftMs, etc.
        //                                juga otherBotsInfo, boardInfo (width, height, allGameObjects), currentDiamondsOnMap

    // Perbarui riwayat posisi dan deteksi stuck dasar
```

```
CALL _update_history_and_simple_stuck_detection(myBotInfo.currentPos,gameState.current_move_command) // Asumsi ada last_move

    // Logika keputusan hierarkis Garox
    // Narasi:
    // Bot Garox pertama-tama mengevaluasi apakah kondisi mendesak untuk kembali ke markasnya.
    // Keputusan ini mempertimbangkan isi inventaris, sisa waktu krusial untuk perjalanan pulang yang aman,
    // atau kondisi mendesak jika waktu permainan hampir habis. Jika diputuskan harus pulang,
    // maka markas (atau teleporter efisien menuju markas) akan menjadi tujuan utama.

    // Jika tidak ada keharusan pulang, Garox akan menilai apakah ada target diamond yang sangat menguntungkan.
    // Penilaian ini kompleks, menghitung skor evaluasi berdasarkan poin diamond, estimasi total waktu
    // untuk mengambil dan kembali ke markas (termasuk penggunaan teleporter), serta faktor urgensi
    // berdasarkan sisa waktu dan seberapa penuh inventarisnya. Diamond dengan skor evaluasi tertinggi akan dipilih.

    // Apabila tidak ada diamond yang cukup menarik, atau jika ada peluang emas, Garox akan mempertimbangkan
    // untuk menyerang (melakukan tackle) pada bot lawan yang terlihat membawa banyak diamond dan berada dalam jangkauan.
    // Ini adalah manuver berisiko tinggi namun berpotensi memberikan imbalan besar.

    // Jika ketiga opsi di atas tidak menghasilkan target, Garox akan mengevaluasi penggunaan Tombol Merah.
    // Ini dilakukan jika bot tidak membawa diamond, jumlah diamond di papan sedikit atau kualitasnya buruk,
    // dan bot berada cukup dekat dengan tombol serta cooldown memungkinkan.

    // Sebagai pilihan terakhir, jika tidak ada aksi strategis yang dapat diambil, Garox akan melakukan
    // gerakan roaming yang aman untuk menjelajahi peta dan menghindari diam di tempat.

    // Setelah target (goal_position) ditentukan dari salah satu langkah di atas, Garox menghitung
    // jalur efektif ke sana. Jika target adalah pintu masuk teleporter, status khusus akan dicatat.
```

```
    // Langkah pertama dari jalur tersebut kemudian dieksekusi.

    // Variabel untuk menyimpan hasil keputusan dari setiap tahap evaluasi
                targetForBaseReturn ← CALL _evaluate_return_to_base(myBotInfo, boardInfo, gameState.allGameObjects)

    IF targetForBaseReturn.shouldReturn THEN
        goal_position ← targetForBaseReturn.targetPosition
        current_target_is_teleporter_entry ← targetForBaseReturn.isTeleporterEntry
    ELSE
            targetFromDiamondEval ← CALL _evaluate_best_diamond_and_path_to_target(myBotInfo, boardInfo, gameState.allGameObjects, currentDiamondsOnMap)
                    targetFromTackleEval ← CALL _evaluate_tackle_opponent(myBotInfo, boardInfo, gameState.allGameObjects, otherBotsInfo)
                    targetFromRedButtonEval ← CALL _evaluate_red_button(myBotInfo, boardInfo, gameState.allGameObjects, currentDiamondsOnMap)

        // Pilih target terbaik dari diamond, tackle, atau tombol merah berdasarkan prioritas/skor
        // (Logika pemilihan prioritas antara diamond, tackle, dan tombol merah perlu didetailkan di sini
        //  berdasarkan deskripsi Garox, misal tackle jika skor diamond rendah tapi tackle bagus)

        // Contoh sederhana: prioritaskan diamond jika skornya tinggi, lalu tackle, lalu tombol
          IF targetFromDiamondEval.targetPosition IS NOT NIL AND targetFromDiamondEval.score > BASE_MIN_TARGET_EVALUATION THEN
            goal_position ← targetFromDiamondEval.targetPosition
            current_target_is_teleporter_entry ← targetFromDiamondEval.isTeleporterEntry
        ELSEIF targetFromTackleEval.targetPosition IS NOT NIL THEN
            goal_position ← targetFromTackleEval.targetPosition
                current_target_is_teleporter_entry ← FALSE // Tackle biasanya bukan via TP entry secara langsung
        ELSEIF targetFromRedButtonEval.targetPosition IS NOT NIL THEN
            goal_position ← targetFromRedButtonEval.targetPosition
            current_target_is_teleporter_entry ← FALSE // Tombol juga bukan TP entry
        ELSE
            goal_position ← NIL // Tidak ada target dari evaluasi, akan roaming
            current_target_is_teleporter_entry ← FALSE
```

```
      ENDIF
   ENDIF

   // Menentukan gerakan final berdasarkan goal_position
   final_dx ← 0
   final_dy ← 0
   IF goal_position IS NOT NIL THEN
         // Jika target saat ini adalah pintu masuk teleporter dan bot sudah di sana, lakukan gerakan acak/spesifik untuk memicu teleport.
                  IF current_target_is_teleporter_entry AND _positions_equal(myBotInfo.currentPos, goal_position) THEN
         final_dx, final_dy ← CALL _get_move_to_trigger_teleport(myBotInfo.currentPos, boardInfo)
           // Setelah ini, current_target_is_teleporter_entry harusnya direset dan goal_position diupdate ke target sebenarnya.
           // Atau, flag ini menandakan bahwa langkah berikutnya harusnya membawa ke tujuan akhir via teleporter.
         // Logika ini penting untuk Garox.
       ELSE
          // Dapatkan langkah berikutnya dari jalur menuju goal_position (jalur sudah dihitung di dalam fungsi evaluasi)
         // Untuk Garox, fungsi evaluasi (diamond, base) sudah mengembalikan path atau next_step.
         // Jika belum, panggil _calculate_effective_distance_and_path lagi di sini jika hanya posisi.
           // Kita asumsikan goal_position adalah next_step jika path sudah diproses, atau target akhir jika belum.
         // Jika goal_position adalah target akhir, kita perlu path.
         // Mari sederhanakan: kita butuh fungsi yang memberi dx,dy ke goal_position
            effectivePathInfo ← CALL _calculate_effective_distance_and_path(myBotInfo.currentPos, goal_position, boardInfo, CALL _get_teleporter_data(boardInfo, gameState.allGameObjects))
         IF effectivePathInfo.path IS NOT EMPTY THEN
                              moveCmd ← CALL getCommandForFirstStep(myBotInfo.currentPos, effectivePathInfo.path)
                  final_dx, final_dy ← CALL _convert_move_command_to_delta(moveCmd) // Fungsi utilitas
            // Update jika langkah pertama adalah pintu teleporter
                                          IF LENGTH(effectivePathInfo.path) > 0 AND IS_TELEPORTER_ENTRY(effectivePathInfo.path[0], gameState.allGameObjects) THEN
```

```
                current_target_is_teleporter_entry ← TRUE
                // Goal_position mungkin perlu diupdate menjadi pintu TP jika belum
                // atau jika effectivePathInfo.isTeleporterUsed adalah true.
            ELSE
                current_target_is_teleporter_entry ← FALSE
            ENDIF
        ELSE // Gagal path
                    final_dx, final_dy ← CALL _get_roaming_or_safe_move(myBotInfo.currentPos,
boardInfo, otherBotsInfo)
        ENDIF
    ENDIF
  ELSE // Tidak ada goal_position, maka roaming
        final_dx, final_dy ← CALL _get_roaming_or_safe_move(myBotInfo.currentPos, boardInfo,
otherBotsInfo)
  ENDIF

  // Update `consecutive_identical_moves` berdasarkan (final_dx, final_dy) dan gerakan sebelumnya.
  CALL _update_consecutive_moves_status(final_dx, final_dy)

  RETURN (final_dx, final_dy)
ENDFUNCTION

// --- Metode-Metode Pembantu Utama untuk Garox (sesuai Tabel 2 di 3.3.1) ---

FUNCTION _get_game_info_from_state(gameState: GameState) : CollectedGameInfo
  // Ekstrak dan kembalikan semua data relevan dari gameState ke dalam satu struktur/objek
  // ...
  RETURN relevant_info
ENDFUNCTION

FUNCTION _calculate_effective_distance_and_path(start_pos: Position, end_pos: Position, board:
Board, teleporter_pairs_data: List) : {distance: int, path: List[Position], uses_teleporter: bool,
teleporter_entry_target: Optional[Position]}
  // Hitung jalur BFS langsung.
  // Untuk setiap pasangan teleporter:
  //   Hitung jalur BFS dari start_pos ke teleporter_masuk_A.
```

```
    //   Hitung jalur BFS dari teleporter_keluar_B (pasangan A) ke end_pos.
    //   Total jarak = jarak_ke_TP_A + jarak_antar_TP (biasanya 1 atau 0) + jarak_dari_TP_B_ke_end.
    // Bandingkan semua jalur efektif (langsung dan via semua teleporter), pilih yang terpendek.
     // Kembalikan jarak, path (daftar posisi), boolean apakah teleporter digunakan, dan posisi pintu masuk teleporter jika digunakan.
    // ... (Implementasi detail penting di sini) ...
    RETURN best_path_info
ENDFUNCTION

FUNCTION _get_teleporter_data(board: Board, allGameObjects: List<GameObject>) : List<Tuple<Position, Position, int>> // Cacheable
    IF teleporter_data_cache IS NOT NIL THEN RETURN teleporter_data_cache
        // Proses allGameObjects untuk menemukan semua TeleporterGameObject, identifikasi pasangannya,
    // dan hitung jarak (langkah) antar pasangan teleporter tersebut (biasanya 1 jika langsung, atau 0).
    // Simpan hasilnya di teleporter_data_cache.
    // ...
    RETURN cached_teleporter_data
ENDFUNCTION

FUNCTION _evaluate_return_to_base(myBotInfo, board, allGameObjects) : {shouldReturn: bool, targetPosition: Optional[Position], isTeleporterEntry: bool}
    // Implementasi logika hierarkis Garox untuk kembali ke base:
    // 1. Jika inventory full (myBotInfo.diamondsHeld >= myBotInfo.inventorySize).
     // 2. Jika waktu kritis: estimasi waktu pulang (via _calculate_effective_distance_and_path ke base) + TIME_SAFETY_MARGIN_MOVES > sisa waktu.
     // 3. Jika urgensi akhir game: sisa waktu < URGENT_TIME_PERCENTAGE * total_waktu_game DAN
      //     inventory >= URGENT_RETURN_THRESHOLD_PERCENT * inventorySize DAN waktu kritis.
    // Jika salah satu terpenuhi dan membawa diamond, hitung jalur efektif ke base.
    // `targetPosition` adalah langkah pertama dari jalur efektif itu atau pintu teleporter.
    // ...
    RETURN decision_info
ENDFUNCTION
```

```
FUNCTION _evaluate_best_diamond_and_path_to_target(myBotInfo, board, allGameObjects, currentDiamondsOnMap) : {targetPosition: Optional[Position], path: List[Position], score: float, isTeleporterEntry: bool}
    // Iterasi semua `currentDiamondsOnMap`.
    // Untuk setiap diamond:
    //   Hitung jalur efektif ke diamond (path1_info = _calculate_effective_distance_and_path).
    //   Jika bisa dijangkau, hitung jalur efektif dari diamond ke base (path2_info).
    //   Estimasi total_time = path1_info.distance + path2_info.distance.
    //   Hitung `current_target_evaluation_score` menggunakan _calculate_diamond_score_heuristic.
    // Pilih diamond dengan skor tertinggi yang juga di atas BASE_MIN_TARGET_EVALUATION.
      // Kembalikan posisi diamond (atau pintu teleporter menuju diamond), path lengkap, skor, dan status teleporter.
    // ...
    RETURN best_diamond_choice_info
ENDFUNCTION

FUNCTION _calculate_diamond_score_heuristic(diamond_points: int, time_to_get_diamond: int, time_from_diamond_to_base: int, time_left_ms: int, inventory_fill_ratio: float, total_game_time_ms: int) : float
    // Implementasi formula skor Garox:
    //   base_score = diamond_points / (time_to_get_diamond + time_from_diamond_to_base + 1)
    //   urgency_factor = 1 + ( (total_game_time_ms - time_left_ms) / total_game_time_ms ) // Semakin mendesak, semakin besar
    //   greed_factor = 1 - inventory_fill_ratio // Semakin kosong inventory, semakin besar
    //   final_score = base_score * urgency_factor * greed_factor
    // ...
    RETURN final_score
ENDFUNCTION

FUNCTION _evaluate_tackle_opponent(myBotInfo, board, allGameObjects, otherBotsInfo) : {targetPosition: Optional[Position], path: List[Position], isTeleporterEntry: bool}
    // Cari otherBotsInfo dalam TACKLE_RADIUS.
    // Pilih lawan yang membawa >= TACKLE_MIN_OPPONENT_DIAMONDS.
    // Pertimbangkan risiko: jika myBotInfo.diamondsHeld juga banyak, mungkin batalkan serangan.
    // Hitung jalur efektif ke lawan. Jika bisa dijangkau dan menguntungkan, set target.
    // ...
```

```
    RETURN tackle_decision_info
ENDFUNCTION

FUNCTION _evaluate_red_button(myBotInfo, board, allGameObjects, currentDiamondsOnMap) : {targetPosition: Optional[Position], path: List[Position], isTeleporterEntry: bool}
                // Cek kondisi: tidak bawa diamond, jumlah diamond di papan < LOW_DIAMOND_COUNT_FOR_RED_BUTTON,
        //                 jarak ke tombol wajar, cooldown (gameState.turn - last_red_button_press_turn > RED_BUTTON_COOLDOWN_TURNS).
    // Jika terpenuhi, hitung jalur efektif ke tombol.
    // ...
    RETURN red_button_decision_info
ENDFUNCTION

FUNCTION _get_roaming_or_safe_move(currentPos: Position, board: Board, otherBotsInfo: List[BotInfo]) : Tuple[dx, dy]
      // Jika stuck (consecutive_identical_moves > STUCK_MOVE_LIMIT), pilih gerakan acak yang valid dan aman dari tackle.
    // Jika tidak, ikuti pola `roaming_directions` jika valid dan aman.
    // ... (Implementasi detail untuk memilih (dx,dy) yang aman)
    RETURN (safe_dx, safe_dy)
ENDFUNCTION

FUNCTION _update_history_and_simple_stuck_detection(currentPos: Position, lastMoveCommand: Tuple[dx,dy])
    // Tambah ke position_history.
    // Update consecutive_identical_moves.
    // ...
ENDFUNCTION

FUNCTION _handle_successful_return_to_base_resets()
    // Reset turns_since_last_diamond_scored.
    // Mungkin reset beberapa status failed_targets jika diinginkan.
ENDFUNCTION

FUNCTION handleUnreachableTarget(unreachableGoal: Position, currentTurn: int)
```

```
    // Jika unreachableGoal adalah diamond, bisa ditambahkan ke semacam daftar failed_path_targets
    // atau logika serupa dengan manageFailedTargetState.
ENDFUNCTION

FUNCTION processGoalReached(gameState: GameState, reachedGoal: Position, basePos: Position, redButtonPos: Optional[Position])
    // Logika setelah mencapai tujuan
    IF reachedGoal = basePos THEN
        CALL _handle_successful_return_to_base_resets()
        just_pressed_red_button_last_turn ← FALSE
        turns_since_last_diamond_scored ← 0
    ELSEIF redButtonPos IS NOT NIL AND reachedGoal = redButtonPos THEN
        last_red_button_press_turn ← gameState.turn
        just_pressed_red_button_last_turn ← TRUE
    ELSE // Sampai di diamond (atau target tackle)
        // Keberhasilan pengambilan diamond atau tackle akan dicek pada giliran berikutnya
        // melalui perubahan gameState.myBot.diamondsHeld atau score.
        // Di sini hanya reset goal, karena sudah sampai.
    ENDIF
    goal_position ← NIL // Siap untuk target baru
    current_target_is_teleporter_entry ← FALSE
ENDFUNCTION

FUNCTION updatePreviousBotStateSnapshot(gameState: GameState, diamondsHeld: int, currentPos: Position, currentActiveGoal: Optional[Position])
        // Simpan status relevan ke prev_bot_state_for_failure_check untuk deteksi progres di giliran berikutnya.
    // ...
ENDFUNCTION

ENDCLASS
```

2. Penjelasan Alur Program

Alur kerja bot yang mengimplementasikan Algoritma G (Garox), yang bersifat agresif dan oportunistik, pada setiap giliran adalah sebagai berikut:

Ekstraksi Informasi dan Update Status Awal: Setiap giliran dimulai dengan bot mengekstrak informasi krusial dari GameState yang diterima. Ini termasuk status internalnya sendiri (posisi, diamond yang dibawa, posisi *base*, sisa waktu), informasi mengenai semua diamond yang tersedia di papan, posisi dan status bot-bot lawan, serta detail objek peta lainnya seperti teleporter dan Tombol Merah. Bersamaan dengan itu, bot juga memperbarui beberapa status internalnya, seperti riwayat posisi untuk deteksi kondisi terjebak dan mengelola *cooldown* untuk Tombol Merah atau target yang sebelumnya diabaikan.

Evaluasi Hierarkis untuk Penentuan Aksi Prioritas: Bot Garox kemudian melakukan serangkaian evaluasi keputusan secara hierarkis untuk menentukan aksi terbaik. Urutan prioritas ini dirancang untuk memaksimalkan peluang dan mengelola risiko secara dinamis:

1. **Prioritas Utama: Keputusan Kembali ke *Base* (Pulang):**

    - Langkah pertama dan paling krusial adalah mengevaluasi apakah bot harus segera kembali ke *base* untuk mengamankan diamond yang sudah dibawa. Keputusan ini dipicu oleh beberapa skenario:
        - **Inventaris Penuh:** Jika jumlah diamond di *inventory* telah mencapai kapasitas maksimal atau ambang batas yang ditentukan (NORMAL_RETURN_THRESHOLD_PERCENT dari kapasitas).
        - **Waktu Kritis untuk Perjalanan Pulang:** Bot menghitung estimasi waktu yang dibutuhkan untuk kembali ke *base* dari posisinya saat ini (mempertimbangkan jalur efektif, termasuk penggunaan teleporter, dan ditambah TIME_SAFETY_MARGIN_MOVES). Jika estimasi waktu ini melebihi atau sangat mendekati sisa waktu permainan, dan bot membawa sejumlah diamond, maka kembali ke *base* menjadi prioritas.
        - **Urgensi Akhir Permainan:** Jika sisa waktu permainan sudah sangat menipis (misalnya, di bawah URGENT_TIME_PERCENTAGE dari total

waktu) dan bot membawa sejumlah diamond minimum tertentu (sesuai URGENT_RETURN_THRESHOLD_PERCENT dari kapasitas), bot akan dipaksa untuk segera kembali ke *base*.

- Jika salah satu kondisi ini terpenuhi, goal_position bot akan diatur ke posisi *base*-nya (atau ke pintu masuk teleporter yang merupakan bagian dari jalur efisien menuju *base*), dan status current_target_is_teleporter_entry akan disesuaikan jika teleporter digunakan.

2. **Opsi Kedua: Evaluasi Diamond Terbaik untuk Diambil:**

    - Jika tidak ada keharusan mendesak untuk kembali ke *base*, bot akan mencari peluang untuk menambah diamond. Ini dilakukan melalui fungsi _evaluate_best_diamond_and_path_to_target. Untuk setiap diamond yang ada di papan:
        - Dihitung jalur efektif (terpendek, mempertimbangkan teleporter) dari posisi bot ke diamond tersebut (time_to_get_diamond).
        - Dihitung juga jalur efektif dari posisi diamond tersebut kembali ke *base* (time_from_diamond_to_base).
        - Total waktu yang dibutuhkan untuk siklus pengambilan diamond dan kembali ke *base* diestimasi.
        - Sebuah skor evaluasi komprehensif (current_target_evaluation_score) dihitung untuk diamond tersebut. Skor ini tidak hanya mempertimbangkan poin diamond dan total waktu, tetapi juga "faktor urgensi" (berdasarkan sisa waktu permainan) dan "faktor keserakahan" (berdasarkan seberapa penuh *inventory* bot saat ini – semakin kosong, semakin besar keinginan untuk mengambil risiko).
        - Diamond dengan skor evaluasi tertinggi, yang juga melebihi ambang batas minimal (BASE_MIN_TARGET_EVALUATION), akan dipilih sebagai target. goal_position akan diatur ke posisi diamond ini (atau pintu masuk teleporter yang relevan).

3. **Opsi Ketiga: Manuver Taktis – Menyerang Lawan (*Tackle*):**

- Jika tidak ada diamond yang dinilai cukup menarik (misalnya, skor evaluasinya rendah) atau jika ada peluang strategis yang lebih baik, Garox akan mempertimbangkan untuk melakukan *tackle* pada bot lawan.
- Bot akan memindai lawan yang berada dalam TACKLE_RADIUS.
- Lawan akan menjadi target potensial jika mereka membawa sejumlah diamond signifikan (minimal TACKLE_MIN_OPPONENT_DIAMONDS).
- Garox juga akan mempertimbangkan kemampuannya untuk mencapai lawan tersebut dan risiko yang mungkin timbul (meskipun dengan bias "full ofensif", risiko ini mungkin lebih ditoleransi).
- Jika target *tackle* yang valid dan menjanjikan ditemukan, goal_position akan diatur ke posisi bot lawan tersebut.

4. **Opsi Keempat: Penggunaan Strategis Tombol Merah:**

   - Apabila tidak ada keputusan untuk kembali ke *base*, tidak ada diamond yang sangat menarik, dan tidak ada peluang *tackle* yang bagus, Garox akan mengevaluasi penggunaan Tombol Merah.
   - Kondisi pemicunya adalah jika bot tidak sedang membawa diamond, jumlah total diamond di papan di bawah ambang batas tertentu (LOW_DIAMOND_COUNT_FOR_RED_BUTTON) yang mengindikasikan kelangkaan, bot berada cukup dekat dengan Tombol Merah, dan periode *cooldown* sejak penggunaan terakhir Tombol Merah (RED_BUTTON_COOLDOWN_TURNS) telah terpenuhi.
   - Jika kondisi ini terpenuhi, goal_position diatur ke posisi Tombol Merah.

5. **Opsi Terakhir: *Roaming* atau Gerakan Aman:**

   - Jika tidak ada satupun dari aksi prioritas di atas yang dapat diambil (misalnya, tidak ada diamond yang layak, tidak perlu ke *base*, tidak ada target *tackle*, tidak ada alasan menekan Tombol Merah, atau gagal menemukan jalur ke target yang sudah ditentukan), bot akan melakukan gerakan *roaming*.
   - Gerakan ini bisa berupa gerakan acak ke sel yang valid dan aman di sekitarnya, atau mengikuti pola tertentu, untuk menghindari kondisi diam dan terus

menjelajahi peta mencari peluang baru. Bot juga akan memperbarui status consecutive_identical_moves untuk mendeteksi jika ia terjebak bahkan saat *roaming*.

Eksekusi Gerakan dan Navigasi Teleporter:

- Setelah goal_position akhir ditentukan dari salah satu langkah evaluasi di atas, bot akan menggunakan fungsi _calculate_effective_distance_and_path (jika belum dilakukan oleh fungsi evaluasi spesifik) untuk mendapatkan jalur terpendek ke sana, yang secara eksplisit mempertimbangkan semua rute teleporter.
- Jika jalur yang dipilih melibatkan penggunaan teleporter, goal_position sementara bot mungkin adalah pintu masuk teleporter, dan status current_target_is_teleporter_entry akan diatur True.
- Ketika bot mencapai pintu masuk teleporter (yang merupakan goal_position-nya saat itu), pada giliran berikutnya ia akan melakukan gerakan tertentu (misalnya, gerakan acak aman di tempat atau "STAY" jika API game engine memproses teleportasi saat berada di atas teleporter) untuk memicu teleportasi. Setelah teleportasi, goal_position akan diperbarui ke target sebenarnya setelah keluar dari teleporter, atau bot akan mengevaluasi ulang dari posisi barunya.
- Langkah pertama dari jalur yang telah dihitung kemudian diambil, dan perintah gerakan yang sesuai dikirimkan ke game engine. Status seperti consecutive_identical_moves dan turns_since_last_diamond_scored diperbarui berdasarkan hasil gerakan.

4.2 Struktur Data yang Digunakan

Struktur data pada atribut dan method yang ada di algoritma greedy optimized :

- **Atribut**

Nama Atribut / Parameter	Tipe Data (Python)	Deskripsi Sederhana dan Kegunaannya dalam Strategi Bot
goal_position	Optional[Position]	Status Dinamis. Koordinat (x,y) tujuan bot saat ini (diamond, base, teleporter masuk, lawan untuk di-tackle, Tombol Merah). Jika None, bot akan roaming atau mencari target baru.
current_target_is_teleporter_entry	bool	Status Dinamis. Bernilai True jika goal_position saat ini adalah pintu masuk teleporter yang ingin digunakan bot sebagai bagian dari jalur menuju target akhir.
position_history	List[Position]	Status Dinamis. Menyimpan beberapa posisi terakhir bot untuk mendeteksi kondisi terjebak (stuck).
last_red_button_press_turn	int	Status Dinamis. Giliran terakhir bot "menekan" Tombol Merah, digunakan untuk logika cooldown.
teleporter_data_cache	Optional[Dict]	Status Dinamis (Cache). Menyimpan data pasangan teleporter yang sudah diproses untuk menghindari kalkulasi ulang pada setiap giliran, mempercepat pencarian jalur efektif.
consecutive_identical_moves	int	Status Dinamis. Menghitung berapa kali bot melakukan gerakan yang sama berturut-turut, indikasi potensi stuck.
turns_since_last_diamond_scored	int	Status Dinamis. Menghitung giliran sejak terakhir kali bot berhasil mengamankan diamond di base, bisa digunakan untuk memicu strategi alternatif jika terlalu lama tidak ada progres skor.
DEFAULT_INVENTORY_SIZE	int	Parameter Konfigurasi. Ukuran standar inventory bot jika tidak ditentukan oleh game.
TIME_SAFETY_MARGIN_MOVES	int	Parameter Konfigurasi. Margin keamanan (dalam jumlah langkah/estimasi waktu) yang ditambahkan saat menghitung kelayakan kembali ke base menjelang akhir permainan.
TACKLE_RADIUS	int	Parameter Konfigurasi. Jarak radius di mana bot akan mempertimbangkan untuk melakukan tackle pada lawan.
TACKLE_MIN_OPPONENT_DIAMONDS	int	Parameter Konfigurasi. Jumlah minimum diamond yang harus dibawa lawan agar dipertimbangkan sebagai target tackle.
OPPONENT_HIGH_DIAMOND_COUNT	int	Parameter Konfigurasi. Jumlah diamond yang dibawa lawan yang dianggap "banyak", bisa memengaruhi keputusan tackle.
LOW_DIAMOND_COUNT_FOR_RED_BUTTON	int	Parameter Konfigurasi. Jumlah diamond minimum di papan yang memicu pertimbangan untuk menekan Tombol Merah (jika diamond terlalu sedikit).
URGENT_TIME_PERCENTAGE	float	Parameter Konfigurasi. Persentase sisa waktu permainan yang dianggap "mendesak", memengaruhi prioritas kembali ke base. (Misal: 0.25 untuk 25% sisa waktu).
NORMAL_RETURN_THRESHOLD_PERCENT	float	Parameter Konfigurasi. Persentase isian inventory yang memicu kembali ke base dalam kondisi waktu normal. (Misal: 0.8 untuk 80% kapasitas).
URGENT_RETURN_THRESHOLD_PERCENT	float	Parameter Konfigurasi. Persentase isian inventory minimum yang memicu kembali ke base dalam kondisi waktu mendesak. (Misal: 0.5

		untuk 50% kapasitas).
BASE_MIN_TARGET_EVALUATION	float	Parameter Konfigurasi. Skor evaluasi target minimum agar sebuah diamond dipertimbangkan untuk diambil.
STUCK_MOVE_LIMIT	int	Parameter Konfigurasi. Jumlah gerakan identik berturut-turut sebelum bot dianggap terjebak dan perlu melakukan gerakan acak.
RED_BUTTON_COOLDOWN_TURNS	int	Parameter Konfigurasi. Jumlah giliran cooldown setelah menekan Tombol Merah.
TIME_PER_STEP_MS_ESTIMATE	int	Parameter Konfigurasi. Estimasi waktu (ms) per langkah, digunakan dalam kalkulasi sisa waktu dan kelayakan aksi.
MIN_TIME_GAIN_FOR_TP_VS_MANHATTAN	int	Parameter Konfigurasi. Minimal selisih waktu (atau langkah) yang harus dihemat agar jalur via teleporter dipilih daripada jalur Manhattan/langsung.

- **Method**

Nama Metode	Deskripsi Sederhana Fungsi dan Perannya dalam Algoritma
__init__(self)	Persiapan Awal Bot. Menginisialisasi semua atribut status internal (misalnya, goal_position, position_history) dan parameter konfigurasi bot ke nilai default atau yang ditentukan. Mempersiapkan bot untuk permainan.
next_move(self, game_state: GameState)	Otak Utama Bot. Metode inti yang dipanggil setiap giliran. Mengorkestrasi seluruh hierarki pengambilan keputusan: evaluasi pulang, cari diamond, serang lawan, tekan tombol merah, atau jalan-jalan. Menghitung jalur dan mengembalikan perintah gerakan (dx, dy).
_get_game_info(self, game_state: GameState)	Pengumpul Informasi. Mengekstrak dan mengorganisir data penting dari GameState (info bot sendiri, info lawan, info papan, daftar diamond, waktu sisa, dll.) agar mudah diakses oleh metode lain.
_calculate_effective_distance_and_path(self, start_pos, end_pos, board, teleporter_pairs)	Kalkulator Jalur Cerdas. Menghitung jarak dan jalur terpendek dari start_pos ke end_pos. Membandingkan jalur langsung (BFS standar) dengan semua kemungkinan jalur melalui setiap pasangan teleporter, lalu memilih yang paling efektif (jarak/waktu tempuh terkecil). Mengembalikan jarak, jalur, dan apakah teleporter digunakan.
_get_teleporter_data(self, board: Board)	Pemroses Teleporter. Mengidentifikasi semua pasangan teleporter aktif di papan dan menyiapkannya dalam format yang mudah digunakan untuk _calculate_effective_distance_and_path. Mungkin menggunakan cache (teleporter_data_cache).

_evaluate_return_to_base(self, my_bot_info, time_left_ms, board, ...)	Logika Pulang. Mengevaluasi semua kondisi yang memicu keputusan untuk kembali ke base (inventaris, waktu kritis, urgensi). Jika harus pulang, mengembalikan posisi base (atau teleporter menuju base) sebagai target.
_evaluate_best_diamond_and_path_to_target(self, my_bot_info, diamonds, ...)	Pemilih Diamond Utama. Mengevaluasi semua diamond yang ada, menghitung skor evaluasi target (current_target_evaluation_score) yang kompleks (poin, waktu ambil+pulang, urgensi, keserakahan), dan memilih diamond terbaik sebagai target. Mengembalikan detail target dan jalurnya.
_calculate_diamond_score_heuristic(self, diamond_points, time_to_get, time_to_return, time_left_ms, ...)	Heuristik Penilaian Diamond. Fungsi inti yang menghitung seberapa "berharga" sebuah diamond berdasarkan berbagai faktor dinamis, bukan hanya poin mentah atau jarak.
_evaluate_tackle_opponent(self, my_bot_info, other_bots_info, board, ...)	Logika Serangan. Mengevaluasi bot lawan di sekitar. Jika ada target tackle yang menguntungkan (bawa banyak diamond, dalam jangkauan, risiko terkendali), mengembalikan posisi lawan sebagai target.
_evaluate_red_button(self, my_bot_info, board, diamonds_on_board, time_left_ms, ...)	Strategi Tombol Merah. Menilai apakah menekan Tombol Merah adalah langkah yang baik (diamond sedikit, bot dekat tombol, cooldown selesai, tidak bawa diamond). Jika ya, mengembalikan posisi tombol sebagai target.
_get_roaming_or_safe_move(self, current_pos, board, ...)	Gerakan Cadangan. Jika tidak ada aksi prioritas lain, bot akan melakukan gerakan acak yang aman (tidak menabrak tembok/lawan) atau mengikuti pola tertentu untuk menjelajah atau menghindari kondisi diam.
_update_history_and_detect_stuck(self, current_pos, last_move)	Anti-Macet. Memperbarui position_history dan consecutive_identical_moves. Mendeteksi jika bot terjebak dan mengembalikan statusnya.
_get_direction_to_target_with_path(self, current_pos, target_pos, path)	Navigasi per Langkah. Mengambil langkah pertama dari path yang sudah dihitung (hasil dari _calculate_effective_distance_and_path) dan mengonversinya menjadi perintah dx, dy.

4.3 Pengujian Program

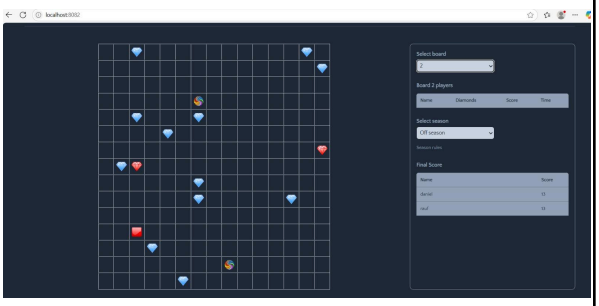
1. Skenario Pengujian

Skenario pengujian pada algoritma ini adalah melakukan 10 kali percobaan dengan rentang waktu 60 detik pada setiap percobaan. Pada percobaan akan digunakan 2 algoritma yang akan ditandingkan satu sama lain, yaitu Algoritma D' dan Algoritma Garox. Algoritma Garox akan dinamai sebagai rauf dan Algoritma D' akan dinamai daniel. Pengujian akan dilakukan terhadap berapa banyak diamond yang dapat dihasilkan oleh masing-masing algoritma.

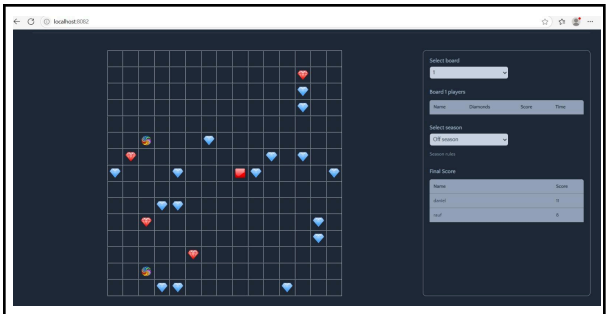
4.4 Hasil Pengujian dan Analisis

match	diamond yang didapatkan	
	Garox	D'
1	13	13
2	8	11
3	14	11
4	10	11
5	10	11
6	10	9
7	10	7
8	12	10
9	11	10
10	8	10
Rata-rata	10.6	10.3

Hasil Match 1 :

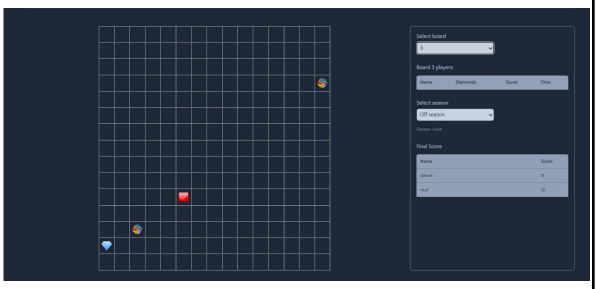


Hasil Match 2 :

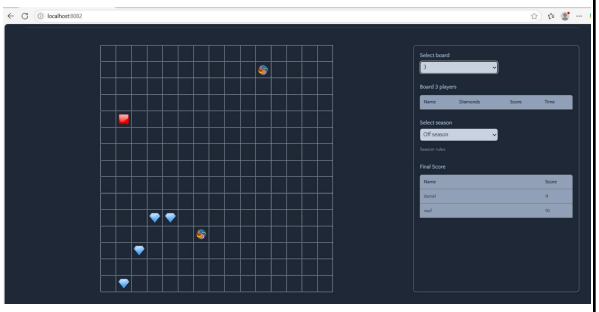


Hasil Match 3 :

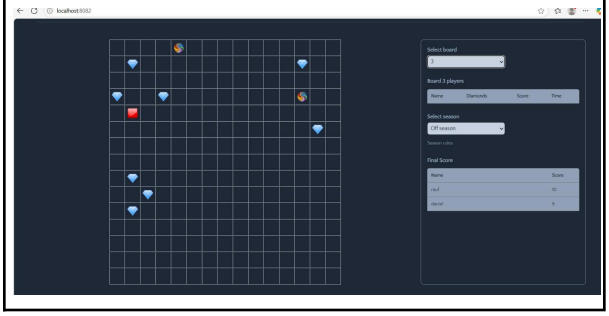
Hasil Match 4 :



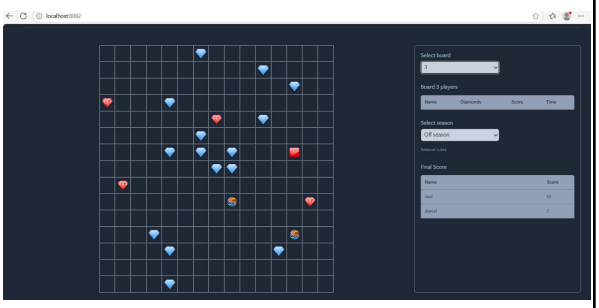
Hasil Match 5 :



Hasil Match 6 :

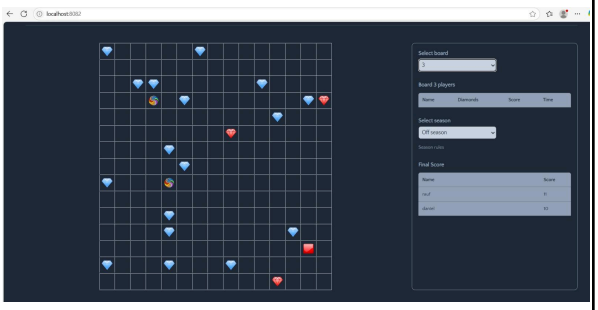


Hasil Match 7 :

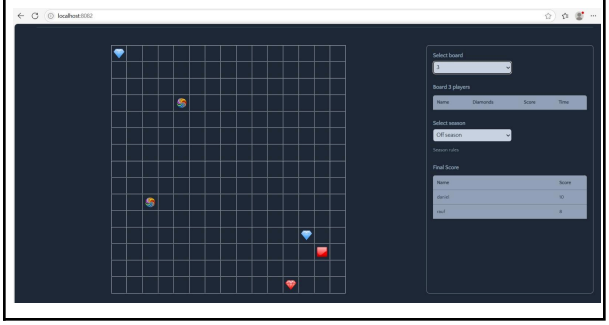


Hasil Match 8 :

Hasil Match 9 :



Hasil Match 10 :



Berdasarkan data hasil pengujian, persaingan antara bot Garox dan bot D' terlihat sengit dan ketat serta kompetitif , berikut hasil analisisnya :

1. **Performa yang Sangat Kompetitif:** Kedua bot menunjukkan performa yang sangat bersaing. Bot Garox memenangkan **5 dari 10 pertandingan**, sementara bot D' memenangkan **4**

pertandingan, dan **1 pertandingan berakhir seri**. Ini menunjukkan bahwa tidak ada dominasi absolut dari satu algoritma terhadap yang lain dalam skenario ini.

2. **Rata-rata Skor yang Berdekatan:** Rata-rata skor yang diperoleh Garox adalah **10.6 diamond** per pertandingan, sedikit lebih tinggi dari rata-rata skor D' yaitu **10.3 diamond**. Selisih rata-rata skor hanya 0.3 diamond, yang secara statistik menunjukkan performa yang hampir seimbang.
3. **Variabilitas Hasil Pertandingan:** Hasil setiap pertandingan menunjukkan variasi yang cukup besar. Ada kalanya Garox unggul dengan selisih 3 diamond (Match 3 & 7), namun ada kalanya D' yang unggul dengan selisih 3 diamond (Match 2). Ini mengindikasikan bahwa faktor-faktor seperti konfigurasi awal peta, kemunculan diamond, dan interaksi spesifik antar bot pada setiap pertandingan memainkan peran penting.
4. **Konsistensi Relatif:**
    - **Garox (Rauf):** Skor berkisar antara 8 hingga 14.
    - **D' (Daniel):** Skor berkisar antara 7 hingga 13.

    Kedua bot menunjukkan rentang skor yang mirip, mengindikasikan bahwa keduanya memiliki kapabilitas dasar yang sebanding dalam mengumpulkan diamond, namun strategi spesifik mereka mungkin unggul dalam kondisi peta atau interaksi tertentu.

Analisis Penyebab Performa yang Kompetitif (Berdasarkan Karakteristik Algoritma):

Performa yang sangat berdekatan ini menunjukkan bahwa kedua algoritma dengan pendekatan yang berbeda, memiliki tingkat efektivitas yang sebanding dalam kondisi pengujian ini.

- **Kekuatan Algoritma Garox:**

    - **Potensi Agresif dan Oportunistik:** Kemampuan Garox untuk mempertimbangkan *tackle* (jika diimplementasikan dan efektif) dan penggunaan Tombol Merah yang strategis memberinya potensi untuk mendapatkan poin tambahan atau mengubah alur permainan. Keunggulan tipis dalam rata-rata skor mungkin berasal dari momen-momen di mana strategi agresif ini berhasil.
    - **Optimalisasi Jalur dengan Teleporter:** Jika peta pengujian memiliki teleporter yang signifikan, kemampuan Garox untuk menghitung jalur efektif bisa memberinya keunggulan dalam efisiensi pergerakan.
- **Kekuatan Algoritma D’ :**

- **Mode Strategi Defensif-Selektif:** Fitur yang memungkinkan bot bermain lebih hati-hati, menghindari lawan, dan fokus pada diamond aman pada fase waktu tertentu bisa jadi sangat efektif dalam mengamankan poin dan menghindari kerugian, terutama jika Garox bermain terlalu agresif dan membuat kesalahan. Ini mungkin menjelaskan mengapa D' bisa memenangkan beberapa pertandingan.
- **Manajemen Risiko yang Baik:** Heuristik seperti CRITICAL_TIME_RETURN_TO_BASE_SECONDS dan pengelolaan target gagal yang terstruktur membantu D' bermain konsisten dan mengamankan poin.
- **Keseimbangan:** Algoritma D’ mungkin menemukan keseimbangan yang baik antara pengumpulan dan pengamanan poin tanpa mengambil risiko sebesar Garox, yang bisa menghasilkan performa stabil.

- **Faktor Lain yang Mungkin Mempengaruhi:**

  - **Konfigurasi Peta:** Tata letak diamond, rintangan, dan teleporter pada setiap peta acak bisa jadi lebih menguntungkan satu algoritma dibandingkan yang lain pada pertandingan tertentu.
  - **Interaksi Langsung:** Bagaimana kedua bot bereaksi terhadap keberadaan satu sama lain (misalnya, jika Garox mencoba *tackle* dan D’ berhasil menghindar atau sebaliknya) akan sangat mempengaruhi hasil.
  - ***Tuning* Parameter:** Keberhasilan kedua bot sangat bergantung pada seberapa baik parameter heuristiknya diatur. Sedikit perubahan pada parameter bisa mengubah perilaku dan efektivitas secara signifikan.

Algoritma Garox dan Algoritma D’ menunjukkan tingkat performa yang sangat kompetitif dan relatif seimbang dalam skenario pengujian yang diberikan. Garox memiliki sedikit keunggulan dalam rata-rata perolehan diamond dan jumlah kemenangan, yang mungkin disebabkan oleh sifatnya yang lebih oportunistik dan potensi strategi ofensifnya. Namun, kemampuan D’ untuk bermain lebih defensif dan selektif pada waktu-waktu tertentu juga terbukti efektif untuk memenangkan pertandingan.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Algoritma greedy, yang efisien melalui pemilihan solusi optimal lokal berdasarkan kondisi saat itu, telah berhasil diterapkan dalam permainan Diamonds dengan implementasi utama "Algoritma Garox". Algoritma Garox dipilih karena pendekatan strategisnya yang komprehensif, agresif, dan oportunistik. Meskipun pengujian menunjukkan Garox memiliki performa kompetitif dan sedikit unggul dibandingkan Algoritma D', faktor keberuntungan tetap signifikan mempengaruhi hasil akhir permainan, di mana salah satu strategi efisien yang juga teridentifikasi adalah pemilihan langkah berdasarkan rata-rata jarak terendah ke diamond dalam radius tertentu dari base.

5.2 Saran

Mengingat besarnya pengaruh faktor keberuntungan, disarankan bagi pengembang strategi bot Diamonds untuk fokus pada upaya mengurangi risiko dari situasi tidak menguntungkan dan memaksimalkan potensi dari situasi yang menguntungkan. Ini termasuk melakukan penyempurnaan parameter pada Algoritma Garox dan meningkatkan logika manajemen risikonya. Untuk pengembangan tugas besar selanjutnya, sebaiknya perancangan tugas besar lebih menekankan kebutuhan akan algoritma greedy secara deterministik daripada faktor keberuntungan sehingga mengurangi dominasi faktor acak dan memungkinkan eksplorasi teknik lanjutan untuk optimasi.

LAMPIRAN

A. Repository Github (link)

https://github.com/dnielclv4free/Tubes1_Kopiyah.git

B. Video Penjelasan

https://youtu.be/jrxiVNbvH68

DAFTAR PUSTAKA

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[2] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*. New York, NY: McGraw-Hill, 2008.

[3] J. Kleinberg and É. Tardos, *Algorithm Design*. Boston, MA: Pearson Education, 2006. [4] A. Levitin, *Introduction to the Design and Analysis of Algorithms*, 3rd ed. Boston, MA: Pearson Education, 2012.

[5] M. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed. Chichester, West Sussex, England: John Wiley & Sons, 2009.

[6] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Upper Saddle River, NJ: Pearson Education, 2020.

[7] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[8] J. B. Kruskal, Jr., "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48-50, Feb. 1956.